AD-A237 498

RL-TR-91-72
Final Technical Report
June 1991

PERSISTENT DATA/ KNOWLEDGE BASE

SRI International

Donovan Hsieh and Teresa Lunt

Rome Laboratory
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700

91-03382

91 6 25 015

This report has been reviewed by the Rome Laboratory Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RL-TR-91-72 has been reviewed and is approved for publication.

APPROVED: *Robert M. Flo*

ROBERT M. FLO
Project Engineer

APPROVED: *Raymond P. Urtz*

RAYMOND P. URTZ, JR.
Technical Director
Directorate of Command & Control

FOR THE COMMANDER: *R. Raposo*

RONALD RAPOSO
Directorate of Plans & Programs

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE June 1991	3. REPORT TYPE AND DATES COVERED Final Oct 88 – Dec 90

4. TITLE AND SUBTITLE PERSISTENT DATA/KNOWLEDGE BASE	5. FUNDING NUMBERS C – F30602-88-C-0142 PE – 62702F PR – 5581 TA – 21 WU – 85
6. AUTHOR(S) Donovan Hsieh, Teresa Lunt	

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) SRI International 333 Ravenswood Avenue Menlo Park CA 94025-3493	8. PERFORMING ORGANIZATION REPORT NUMBER SRI Project 6888

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Rome Laboratory (COAC) Griffiss AFB NY 13441-5700	10. SPONSORING/MONITORING AGENCY REPORT NUMBER RL-TR-91-72

11. SUPPLEMENTARY NOTES
Rome Laboratory Project Engineer: Robert M. Flo/COAC/(315)330-2805

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

The Persistent Data/Knowledge Base (PDKB) project is a research effort funded by Rome Laboratory and conducted at the Computer Science Laboratory of SRI International. The main goal of this effort was to design/develop persistent object storage techniques capable of handling multiple data objects in a distributed information processing environment. The objective of this effort was to perform an investigation and evaluation and to develop and verify a top level system/segment design specification for a persistent object-oriented data/knowledge base system using techniques for (1) reliable and efficient access to data/knowledge bases, (2) increased programmer/user productivity, and (3) integrated hyper-media data types. Techniques that were investigated include AI/DB workstations, object representation including multimedia databases, temporal change management, persistent object storage media, spatial databases, and massive memory data/knowledge systems. The contribution is a highly intelligent technique for managing very large main memory or traditional secondary storage domains in a multimedia information system environment.

14. SUBJECT TERMS Database technology, knowledge base technology, persistent storage, object-oriented technology, hyper-media data	15. NUMBER OF PAGES 76
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT U/L

Contents

i

Acknowledgments

Chapter 1

Introduction

1.1 Objective

The Persistent Data/Knowledge Base (PDKB) project is a research effort currently conducted at the Computer Science Laboratory of SRI International, Menlo Park California. The objective of this research is to

- Perform an investigation and evaluation of a persistent object-oriented data/knowledge base system.

- Develop and verify its high-level system design specification.

- Explore new concepts and ideas for tight coupling of both data and knowledge in a unified framework.

- Explore innovative concepts and techniques to provide reliable and efficient access to a data-and-knowledge base.

- Provide integrated building blocks for hypermedia information, such as satellite images, bitmaps, and multidimensional spatial data.

1.2 Overview

This report summarizes the results of this research project. In essence, the PDKB project adopts a tight-coupling strategy to couple both data and

knowledge in a unified framework. The PDKB system uses knowledge representation based on a semantic network to encode knowledge. *Knowledge* is defined as abstractions at a higher level than the real-world data that are stored persistently in an object-oriented database (OODB) system.

To capture the semantic-rich knowledge representation in semantic networks, we propose a set of formalized axioms using Prolog-like expressions to extend the modeling capabilities of OODBs that will capture the real-world data semantics and their relationships. These formalized abstractions include *generalization/specialization, aggregation, composition, association, grouping,* and their interactions. These formalized axioms serve as fundamental building blocks for the underlying inferencing engine. Other logical rules and constraints can be further defined by knowledge engineers, database schema designers, and end users using Prolog-like expressions; once defined they are used to encode domain and application-specific knowledge other than the system's inferencing axioms already defined. The inference and database engine are tightly coupled since both knowledge and the real-world data are stored in the same domain, unlike conventional approaches, in which data and knowledge are stored in separate domains.

Part of this research is motivated from work by others on Datalog [2] Semantic Data Models (SDM) [3] [4] [5], and Extended Entity-Relationship Data Models [6]. The fundamental difference distinguishing PDKB from these models is that PDKB provides both semantic organization principles and formalized deductive axioms in the same framework. Datalog lacks the ability to express real-world semantic abstractions, and the SDM lacks formal representation of the model. Many existing concepts and techniques from Datalog and SDM can be directly applied to the PDKB implementation. New concepts such as the use of high-order predicates are also proposed in this report as future research topics.

We developed a prototype system based on the PDKB concept to demonstrate its feasibility. For fast prototyping, we designed a customized database to store real-world domain data in the PDKB Prototype to provide persistence and continuity for both knowledge and data. In the PDKB Prototype, knowledge and real-world data are tightly coupled because of their unified representation and framework, namely, the consistent mapping between the semantic network and object-oriented data models. In unifying these two representations, we also uncovered and resolved certain discrepancies between them, such as the discrepancy in representing physical entities in semantic

networks versus object instantiations in object-oriented models.

We have chosen to implement a customized database for the PDKB Prototype because of limited resources and commercially available object-oriented databases running on Symbolics machines. The current PDKB Prototype customized database provides primitive yet adequate functions for storing complex persistent data objects such as satellite images, geographical maps, and other multimedia spatial information. To facilitate the smooth migration of this customized database to other commercial object-oriented database systems in the future, we have designed a generic calling interface that isolates target-platform system-dependent calls. This will minimize the migration effort and cost.

For the implementation, we have decided to develop the PDKB Prototype on Lisp machines for prototyping efficiency and performance considerations. In the future, we may port the PDKB Prototype from Symbolic Lisp machines to the Lisp environment on SUN workstations. We also plan to reimplement the PDKB Prototype with other high-level object-oriented programming languages such as C++, if further funding should become available.

At the top level of the PDKB Prototype is a multimedia application which stores and displays bitmap images for a three-dimensional map, taking advantage of the underlying data and knowledge base. The user may browse the bitmap image display with the multimedia interface, or may use the PDKB Prototype query language to reason with the underlying knowledge.

1.3 Future Work

We foresee three alternatives ways in which this project could become a full-fledged system:

- Collaborate with third parties and other research institutions to integrate existing OODB systems, and participate with the current OODB community to standardize the OODB architecture and interfaces.

- Develop a generic interface to couple with off-the-shelf OODBs.

- Implement an in-house persistent-object manager that will provide necessary storage and access mechanism for the PDKB model.

The last approach appears to be least attractive as it requires significant resources and effort. The second approach, which uses off-the-shelf OODBs, has the advantage that the PDKB will include a generic interface which is portable across various OODBs products. Unfortunately, however, the approach loses the tight-coupling feature because PDKB is treated as a regular application. This disadvantage could be avoided only if the PDKB were coupled within the kernel of a commercial OODB product.

The first approach, collaboration, appears to be most attractive; however, it is constrained by the availability and schedule of other parties. We have started pursuing the first two approaches. We have evaluated various commercial OODB products such as Gemstone from Servio Logic, Ontos from Ontologic, Itasca (a commercial version of Orion OODBs from MCC), and a persistent object-storage manager from Xidak. We are also collaborating with other research institutions that are building Open OODB architectures based on community consensus.

1.4 Report Organization

Chapter 2 reviews various ways in AI to encode and represent knowledge. The representations included are those based on semantic networks, object-attribute-value triplets, rules, frames, and logic. Chapter 3 reviews existing database technologies such as relational, object-oriented, entity-relationship, and semantic data models. Chapter 4 surveys relevant technologies and approaches in coupling knowledge-based systems with database systems. Chapter 5 explains why PDKB chooses to tightly couple semantic-network-based knowledge representation with OODBs. Chapter 6 presents the formal framework for the PDKB model. The framework includes meta-level inference axioms and user-defined inference rules using Prolog-like expressions. Chapter 7 describes how queries are processed and rules are inferenced in PDKB. Chapter 8 discusses future research topics and directions for PDKB. They include reasoning from high-order logic object-oriented languages, natural-language processing, machine learning, discretionary and multilevel system security, memory-resident OODB, and characterizations of other semantic abstractions. Chapter 9 concludes this report with the contribution of this research.

Chapter 2

Knowledge Representation in AI

A knowledge-based system is an intelligent computer program that uses knowledge and inference procedures to solve problems that are difficult enough to require significant human expertise for their solution. Knowledge necessary to perform at such a level, plus the inference procedures used, can be thought of as a model of the expertise of the best practitioners of the field.

The knowledge of a knowledge-based system consists of facts and heuristics. The facts may constitute a body of information that is widely shared, publicly available, and generally agreed upon by experts in a field, or they may consist of externally collected data or observations. The heuristics are mostly rules of good judgment (rules of plausible reasoning, rules of good guessing) that characterize expert-level decision making in the field. A knowledge-based system often considers a number of competing hypotheses simultaneously, and frequently makes tentative recommendations or assigns weights to alternatives. The performance level of a knowledge-based system is primarily a function of the size and the quality of the knowledge base it possesses.

In early knowledge-based systems, most of the effort was devoted to the representation of knowledge and the reasoning engine. Typically, knowledge and data stored in these systems are in separate domains. For example, a knowledge system session usually starts with initializing the knowledge base from its file system, entering assertions (base facts), loading input files or data from a network for reasoning, analyzing the result, and refining the

knowledge base before the next session. With a large knowledge base and a complex problem, this approach becomes cumbersome and time consuming. Further more, the knowledge becomes very difficult for multiple users to maintain and track.

Historically, five different approaches have been used to encode the facts and relationships that constitute knowledge. Each method has advantages and disadvantages. These are discussed below.

2.1   Knowledge Representation Based on Semantic Networks

The semantic network is the most representational scheme, and also one of the oldest. A semantic network is a collection of objects called *nodes*. The nodes are connected by arcs or links. Ordinarily, both the links and the nodes are labeled. There are no absolute constraints as to how nodes and links are named, although some conventions can be applied [7]. Flexibility is a major advantage of this representation scheme. New nodes and links can be defined as needed. Inheritance is another feature of semantic networks. It refers to the ability of one node to inherit characteristics and properties of other nodes related to it.

Systems based on this approach use nodes and arcs (or links) to represent knowledge. Although no absolute constraints exist as to how nodes and links are named in semantic networks, there are some conventions, suggested as follows:

- Nodes are used to represent *objects* and *descriptors*. Objects are essentially the same as in object-oriented models except that they also represent instances of object, which is a separate notion in object-oriented models. Objects may be physical such as vehicles or buildings, or may be conceptual entities such as acts, events, or abstract entities. Descriptors provide additional information about objects. In object-oriented models, a description of an object is typically defined as attributes of the object class.

- Links relate objects and descriptors. A link may represent an arbitrary relationship. Common links include the following:

- *IS_A* links are often used to represent the class/instance relationship. In object-oriented models, *IS_A* links denote specialization or inheritance between class and subclass.

- A second common relationship is the *HAS_A* link. *HAS_A* links identify nodes that are properties of other nodes. Typically *HAS_A* links show part-subpart relationships. *HAS_A* links are essentially the same as the *IS_PART_OF* links in object-oriented models to represent the composition abstraction.

- Some links are definitional, or heuristic. Heuristic relationships enrich the network by providing additional paths and semantics. In object-oriented models, arbitrary links of this type are called *association* links. We will discuss the implication on the mapping of such links from knowledge-base systems to object-oriented systems further.

Semantic networks provide flexibility to encode new knowledge into an existing knowledge base by adding new arcs and links to current networks. They also allow one node to inherit characteristics of other nodes that are related to it. This is essentially the same as in object-oriented models. Some systems, such as those of Chow [34] and Moad [35], use a frame-based representation to store knowledge. These systems can be categorized as subsets of semantic networks. Also, representations based on object-attribute-value triplets can be considered as a special case of both semantic networks and frames.

When reasoning with semantic network systems, the inference engine relies on traversing a network based on types (or descriptions) of arcs (or links) and nodes connected. It is important that such systems use efficient graph-traversing schemes to provide efficient inferencing performance when dealing with a large body of knowledge. The semantic networks offers a natural mapping to object-oriented database models since both systems use similar abstractions in capturing real-world semantics However specific modeling discrepancies have to be resolved in order to couple these two systems tightly together. Figure 2.1 illustrates the point with a simplified detective's semantic-network knowledge base.

In the example of Figure 2.1, three fundamental discrepancies separate semantic networks in a knowledge-based system from object-oriented models:
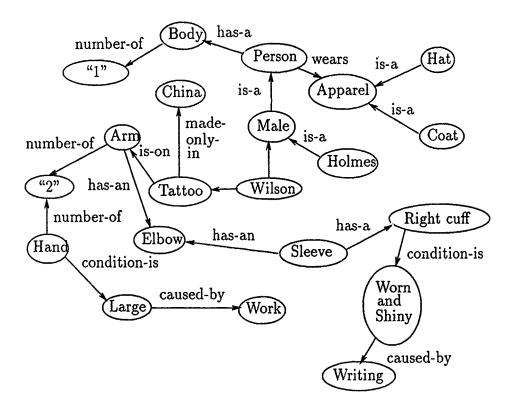
Figure 2.1: A detective's knowledge base represented in a semantic network

- Semantic networks treat object instances as regular objects while object-oriented models treat object instances as separate notions. In our detective example, the node *Wilson* who is a male is an instance of *Male* object.

- Some definitional or heuristic links in semantic networks can be represented as attributes of objects rather than links required in object-oriented models. For example if we were to represent the information that *Wilson's Tattoo* was made in *China* in an object-oriented model, we would define an attribute *"Place_Made"* within the *Tattoo* object class rather than drawing an explicit link between them. Note that the attribute of an object is similar to the slot defined in the frame representation.

- Semantic networks typically employ large number and different types of links that are not directly and formally supported in object-oriented models[1]. Some database researchers [32] [41] [33] have proposed using the notion of *association* to represent arbitrary uscr-defined relationships among objects. However, association provides relatively weak semantic constraints compared to the well-understood generalization abstractions.

To tightly couple knowledge-base systems using semantic networks with object-oriented databases at the data modeling level, we envision a scaled-down representation of the semantic networks and enhancement of the semantic data-modeling capabilities in the object-oriented models. We also need to formalize the semantics of various links in object-oriented data model to match with the richer data-modeling capability of semantic networks.

The interaction between those two tightly coupled systems can be described as follows:

- Both inference engine and database engine will share same kernel services, i.e., knowledge base and real-world data are stored using the same framework with the same representation.

---

[1]Generalization (or inheritance) link is the only well defined and accepted semantic abstraction in object-oriented models.

- The knowledge acquisition and management process of the inference controller will use the schema management service provided by the database controller, i.e., the knowledge base becomes the database schema and run-time instances of the schemata. Database controllers that offer schema versioning capability, such as Orion from MCC [19], should also allow reasoning from different knowledge viewpoints with different temporal factors.

- The interaction enables concurrent reasoning sessions on the same knowledge base with the ability to recover and roll back or roll forward from incomplete transactions.

- The indexing, clustering, and replication features provided by the database controller enable the inference controller to handle a large volume of knowledge and a large data base without having to cache all of the entries into main memory at the beginning of a reasoning session.

- Database queries are supplemented with the inferencing capability, to allow intelligent query processing through embedded knowledge of the data representations and their relationships.

As will be shown, there are different techniques and alternatives for housing the knowledge-based and database systems under the same framework. It is important for a tightly coupled knowledge-base and database system to be able to interface with external or foreign database systems, as large volumes of data reside on heterogeneous database platforms. We foresee our system as providing options either to retrieve data from foreign databases with certain standard data exchange formats, or to acquire and import foreign data into the native system platform for a more efficient inferencing capability.

2.2   Knowledge Representation Based on Object-Attribute-Value Triplets

Another common way to represent factual information is as object-attribute-value (O-A-V) triplets. In this scheme, objects may be physical entities such as a door or a transistor, or they may be conceptual entities such as

a bank loan or a sales episode. Attributes are general characteristics or properties associated with objects. Size, shape, and color are typical attributes for physical objects; interest rate is an attribute for a bank loan. The value of the triplet specifies the nature of an attribute in a particular situation. Representing knowledge with O-A-V triplets is a specialized case of the semantic-network approach. Exotic links are banished in favor of just two simple relationships: the object-attribute link and the attribute-value link. This O-A-V scheme has three features. The first is its distinction between a static, unchanging object and a dynamic instance of that object that changes from case to case. This feature allows a knowledge-based system to store static knowledge "between consultations" with bindings to specific values. As values are determined, the system stores this dynamic information in the working memory. This process of determining specific values for the attributes in a static knowledge base is called instantiation. The second feature of O-A-V representation is that objects are ordered and related to one another. This allows the O-A-V scheme to be fitted conveniently within the semantic network. The third feature of the O-A-V scheme is its ability to handle uncertainty. We can associate a confidence factor (or certainty factor) with the attribute-value pair to represent the confidence that we have in a piece of evidence.

2.3 Rule-Based Knowledge Representation

Rules are widely used to represent relationships. They can be used with either O-A-V or semantic-network representations. A rule is composed of a set of expressions ("if" clauses) that form the premise, plus the conclusion (the "then" clause). Rules link the values for attributes of objects. Rules can be inferenced with forward chaining, with backward chaining, or with other inference strategies. In backward-chaining inference, the system is provided with a specific clause called a goal that is to be proved. To prove a goal, backward-chaining inference begins with and focuses on the conditions of rules. Forward-chaining is the obverse of backward chaining, it focuses on the premises of rules rather than on their conclusions. If the clauses in the premise of a rule are proved, the conclusion of the rule is added to the fact base. Uncertainty factors can also be attached to rules to show the confidence factor of a certain rule deduction.

To contrast rule-based representation with other approaches discussed above, consider the following rule representing a specific fact about our detective knowledge-base example:

```
if    the right cuff of a person's coat sleeve is worn and shiny
      then there is a 60% suggestive evidence that the profession of
           that coat owner is a writer
```

To assert this fact into a knowledge-base system, we could analyze this rule with the following representation:

-	Attribute	Object	Value	Confidence Factor
if	right cuff	coat sleeve	worn and shiny	-
then	profession	coat owner	writer	60%

To encode this fact into a knowledge-base database system, we need to convert the rule description into a well-defined persistent representation. We could encode the information using semantic networks as shown in Figure 2.1 based on their object, attribute, and value representations. We may need to further divide *coat sleeve* and *coat owner* into separate objects such as *coat*, *sleeve*, and *owner* if there is a need to make them distinct.

In most current expert systems, rules are encoded into arbitrary internal representations that cannot be accessed by other inference and database engines. Nor do the representations provide object persistence and concurrent access by multiple users; each user operates within his/her knowledge domain. This limitation typically inhibits the ability of those systems to deal with a large knowledge domain in a multiuser environment.

2.4 Frame-Based Knowledge Representation

Frames provide another method for representing facts and relationships. A frame is a description of an object that contains slots for all of the information associated with the object. Slots, like attributes, may store values, pointers to other frames, sets of rules, or procedures by which values may be obtained. The inclusion of these additional features makes frames different from O-A-V triplets. Frames provide richer representations of knowledge. They are also more difficult to develop.

2.5 Logic-Based Knowledge Representation

Propositional logic is a common logical system. Predicate calculus is an extension to propositional logic that takes an object as elementary unit. Statements about objects are called predicates. Logical formulations represent knowledge in a manner different from the preceding approaches. In logic, users can only ask questions, and those questions can only return either true or false answers.
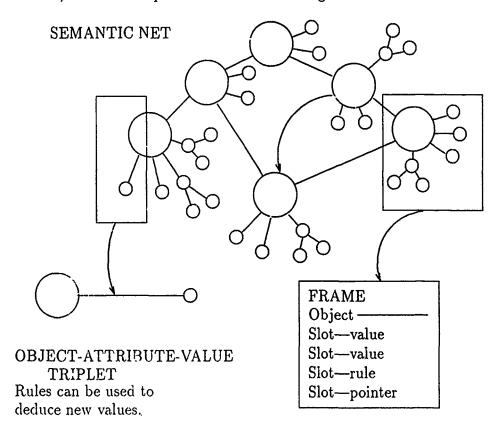
The idea of using first-order logic to describe real-world semantics is not new. For deductive databases [31] [30], logic has been used as the data definition language based on the relational data model. It is generally agreed that logic schemes can provide the following advantages [33]:

- The ability of inference rules in terms of which one can define proof procedures.

- A clean, well understood, and accepted formal semantics.

- A natural way to assert queries.

An important drawback of logic schemes however, is the lack of organizational principles for the facts that constitute a database [33]. A second drawback is the difficulty of representing procedural knowledge using its logical representation. To tightly couple a knowledge-base system of this type with database systems requires a seamless integration between logic and semantic databases: i.e., logical representation must provide a formal representation

for semantic data models, and the semantic data models to provide the organization principles for logic representation. The result should further couple with the procedural semantics provided by the logic programming language, such as Prolog [29]. With this approach, procedures can be included as integral parts of our conceptual schemes, and logic can provide a query language to query the semantic database.

Figure 2.2 shows the overall relationships between semantic networks, O-A-V, and frame representations of knowledge.

SEMANTIC NET

OBJECT-ATTRIBUTE-VALUE
   TRIPLET
Rules can be used to
deduce new values.

FRAME
Object ————————
Slot—value
Slot—value
Slot—rule
Slot—pointer

Rules and pointers can be
incorporated directly into
the frame

Figure 2.2: Semantic Nets, O-A-V Triplets, and Frames

Chapter 3

Survey of Current Database Technologies

We begin this examination of current database technologies by surveying various database models, such as relational, object-oriented, entity-relationship, and semantic data models. We compare their data modeling powers from the perspective of knowledge representation. In the comparison, we discuss some of the most frec ently used real-world abstractions, such as *generalization*, *aggregation*, *composition*, *association*, and *grouping*. Some of these abstractions have been added to the original ER model by later researchers. We will show the modeling differences between the relational, ER and object-oriented models in this respect, and will then propose a new extension to the existing object-oriented data model that will capture richer semantics and the relationships between these abstractions.

3.1 Database Models

A data model of a database system is a mathematical formalism with two parts: a notation for describing data and a set of operations used to manipulate the data. Most data models used today are *strictly typed* [14]; i.e., each datum must belong to some category. Data that do not fall into a category have either to be subverted to fall into one, or they cannot be handled and represented in the data model. In addition, some data models assume that the allowable categories are predefined and cannot evolve; the relational data
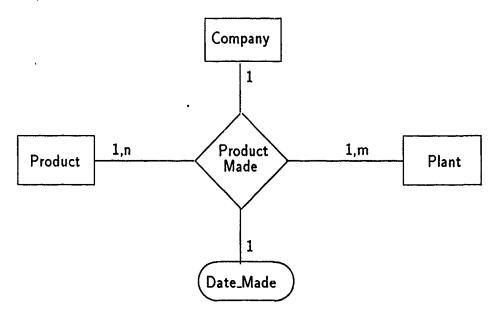
15

model is a typical example. One advantage of this rigidity is that properties of data can be abstracted and investigated in terms of its category. In other words, a theory can be formulated based on the properties of the category. In the relational data model, the relational algebra and relational calculus form the theoretical basis of the relational database system. However, this type of data model poses rigid restrictions on the membership of data in a category. This restriction limits some real-world applications that can be used effectively on top of the strictly typed data model. For example, the relational model is known to be a poor fit for applications such as CASE, CAD/CAM, and multimedia information.

On the other hand, a *loosely typed* data model does not make any assumptions about categories. Categories are allowed only to the extent that they are useful. Individual data can exist by themselves or in relation to some other data. Information about categories, if they exist, is treated in the same way as information about an individual datum. A data-and-knowledge-based system would fall into this category.
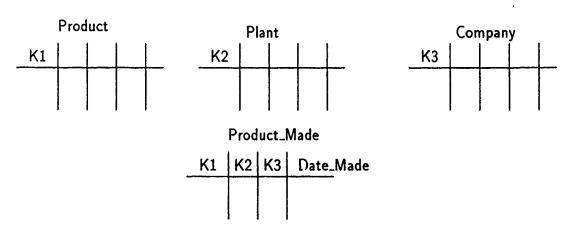
It is difficult to claim that one data model is better than another. For example, the relational model is well suited for traditional MIS data processing, but falls short in supporting other application areas, as mentioned above. Generally speaking, new data models tend to improve on or encompass old data models in terms of modeling capabilities and expressiveness. For example, a relational model provides a more powerful *ad hoc* querying capability than a network data model, and an object-oriented data model provides generalization (or inheritance) and abstract data type capabilities that the relational data model lacks.

For designing database applications, the entity-relationship (ER) data model, based on tables and graphs, has gained wide acceptance in designing the logical database representation. The ER data model was conceived to facilitate conceptual and logical database design by allowing the specification of an *enterprise schema*, which represents the entire enterprise's view of data and is independent of physical database implementation details such as storage or efficiency considerations. The enterprise schema can be further mapped into various physical database models, such as the network, relational and object-oriented models. For example, researchers have proposed algorithms to translate ER diagrams into normalized relational normal forms [15]. Figure 3.1 illustrates the translation from a ternary ER model to relational tables that are in third normal form (3NF).

Entity-Relationship Model :

Relational Model (3NF) :

Product_Made

K1 | K2 | K3 | Date_Made

FD : K1 K2 Date_Made→ K3

FD : K1 K2 K3→ Date_Made

Figure 3.1: Entity-Relationship and Relational Model

3.2   Real-World Semantic Abstractions

The original ER model proposed by Chen [16] uses the concepts of entity type
and relationship type as its basic structures. An entity type is called an *entity
set* and represents the generic structures of an entity in an enterprise's realm
of interest. A relationship type is called a *relationship set*. It represents the
generic structure of the relationship among entity sets. For example, in the
real world there exists a *citizen_of* relationship between *Person* and *country*
entities. In general, any two entities can exist in one or many relationships.
For example, there may exist another relationship *travel_in* between entities
person and *country*.

Generalization and specialization are special kinds of relationships. For
example, the *person* entity may be *specialized* into different entities such as
engineer, architect, accountant, and *doctor* based on its professional classifi-
cation. But an *engineer* entity is still a *person* entity. This is also referred
to as an *IS_A* relationship in the extended ER model. We say that a *person*
entity is a *generalization* of its specialized entities, e.g, *engineer*.

Aggregation is introduced as an abstraction that transforms a relationship
between entities into higher-level entities. For example, a certain relationship
between a person, a hotel, a room, and a date may be abstracted as the
aggregate entity *reservation*. This *reservation* entity may be used without
explicitly specifying all underlying details and relationships.

Composition is an abstraction similar to aggregation. It also transforms a
relationship between entities into higher-level entities. However it emphasizes
the physical structure rather than the abstract structure. The parts hierarchy
of an automobile engine design is a typical example of the composition ab-
straction. Some database models use aggregation for both representations,
but the difference between them are significant enough to justify different
terms. In its newly revised composite object definition [42], Orion also dis-
tinguishes two types of references from one object to another: *weak* and
composite. The weak reference is similar to our aggregation abstraction; the
composite reference has a similar concept except that Orion proposed a set
of formalized semantics through its internal implementation. Composition
abstraction is sometimes called *PART_OF* in some models.

Specific semantics and constraints govern the interactions between various
abstractions. Smith and Smith [17] wrote a pioneering work formalizing the
generalization and aggregation abstractions from the context of the relational

data model. However, both abstractions were treated orthogonally because of the inherent semantic limitation of the relational model. In the object-oriented data model, generalization and specialization are also referred to as *inheritance*. The inheritance in the object-oriented paradigm carries more semantic meaning than in the ER model. It also allows specialized objects to inherit both attributes and methods from its generalization objects.

In general, an object-oriented database (OODB) schema is expressed through its inheritance hierarchy. This is compatible with the inheritance feature in most object-oriented programming languages such as C++. Most OODB systems support composition as an abstraction explicitly or implicitly. If an OODB supports an explicit composition abstraction, it typically provides a set of semantic constraints and high-level operators that allow users to manipulate the composite objects directly. If an OODB supports implicit composition abstraction, users need to define and implement the constraints and operations. Doing so may require a major effort and could potentially affect the behavior of the overall database schema. However, many existing commercial and research OODBs belong to this category.

3.3 Deficiencies in Current Object-Oriented Data Models

To model real-world semantics using current object-oriented data models, database designers are limited to using only the generalization and a limited form of aggregation and composition abstractions, if they are provided by the system. For example, consider the sample automobile-manufacturing database schema used in a recently published Orion paper by Kim [19] in Figure 3.2. This schema shows two types of object relationships that were captured in the object-oriented data model used in ORION: the class/subclass link, which is a generalization abstraction, and the attribute/domain link, which is an aggregation abstraction (note that in the most recent version of Orion, attribute/domain links have been expanded into weak and composite links for composite objects, as explained earlier).

This example shows that the relationship between a *Vehicle* and a *Company* is expressed through an aggregation attribute, *manufacturer*, defined in the *Vehicle* class. Basically, this attribute is a pointer to class *Company*. An
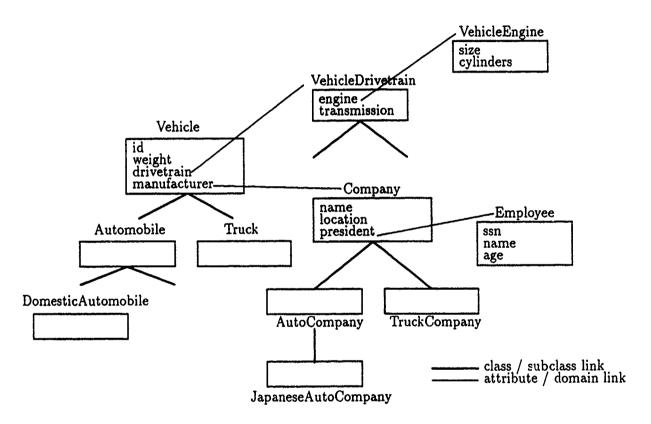
Figure 3.2: Class Hierarchy and Class-Attribute Graph

equivalent representation in the ER model would create an explicit relationship called *Manufacturer_Of* between the *Vehicle* and *Company* entity with many-to-one cardinality from *Vehicle* to *Company*, i.e., a given company may manufacture multiple types of vehicles, but a given vehicle can be manufactured by only one manufacturer. This shows that a straightforward binary relationship in the ER model is actually being treated and interpreted as an aggregation in the object-oriented data model.

Cardinalities associated with a relationship may further complicate the modeling requirements in the object-oriented data model. For example, suppose we add another object class *Factory_Plant* into the above schema to represent the vehicle manufacturing-plant relationships with both *Vehicle* and *Company* object classes: a given vehicle could be manufactured in multiple plants by one company. This poses a difficulty, however, because the object-oriented data model is unable to express the new relationship using the reference from *Vehicle* to *Company*. A possible solution is to decompose the three-way relationship into three binary pairs of aggregations: *Vehicle Company*, *Company Factory_Plant*, and *Vehicle Factory_Plant*. If the underlying OODB does not provide mechanisms to enforce the cardinality constraints, then the user applications must enforce them. This schema decomposition also appears to be rather awkward, since representing the three-way relationship as three pairs of disjointed aggregations does not convey their ternary interrelations in the object-oriented database schema.

The ER model of Teory [15] proposes a translation algorithm that will generate normalized relational tables from n-ary relationships. The cardinality constraint can be expressed using both referential integrity and foreign keys in the relational database schema definition. However, it is not as intuitive and natural to express the same semantics using references in the object-oriented data model. The difficulty arises mainly because of the explicit casting of real-world relationships into the aggregation abstraction by the object-oriented data model.

Another modeling issue relates to the interaction of different abstractions in the object-oriented data model. For example, it is desirable for the *DomesticAutomobile*, a subclass of *Vehicle*, to inherit those composition properties associated with *vehicle*, e.g., each domestically made vehicle must be composed of a *VehicleDrivetrain* and *VehicleEngine*, just as it would inherit those attributes and methods defined in its superclass.

Some subtleties occur when mixing aggregation, composition, and gener-

alization links from a root class to a leaf class. For example, *AutoCompany*, a specialized class of *Company*, exists in the same path from the aggregation link rooted in *Vehicle*. Should this property also be inherited by the *DomesticAutomobile*?

3.4   Findings and Recommendations

The current object-oriented data model, although providing a significant improvement over conventional database models with its ability to inherit superclasses properties, also presents certain modeling deficiencies. Two alternatives would improve the object-oriented data modeling capabilities. The first alternative is to investigate specific deficiencies and shortcomings in the existing model, and to enhance them with specific provisions. For example, we could further explore the interaction semantics between aggregation, composition, and generalization to define suitable policies that could be used and enforced. A variation would be to follow the example of Orion and define object and schema version management semantic constraints as an extension to the object-oriented model. The disadvantage of this approach is that individual enhancements and extensions are still bound to the original modeling limitations; also, introducing extraneous constraints may affect the consistency and integrity of the original model.

The other alternative is to develop an extension of current object-oriented data models with formalized axioms and properties that will provide a sound and complete foundation for various real-world semantic abstractions. If we look at the current object-oriented data models from a different perspective, what we see is basically a directed graph of three link types, generalization, composition, and aggregation, that have specific semantic properties attached to them, e.g., inheritance for generalization, and less-well understood links for aggregation and composition. The basic idea is to develop a generalized *annotated* directed graph model such that the existing object-oriented data model becomes a subset. This new extensions will allow database schema designers to define arbitrary types of link relationships with associated semantics and their cross link constraints. For example, we may wish to define a new link relationship called "configuration," which would define the threading of a set of logically related composite objects. This notion is useful in engineering design applications such as CAD/CAM. We should also be able

to define its semantic constraints, such as the constraint that all levels of versioned objects from the root to the leaf must be included in the composition hierarchy.

3.5 Extending Existing Object-Oriented Data Models

In this section, we describe our extension to the current OODB model. Informally, an OODB is a collection of *entity objects* and *association objects* communicating via *messages*. Both entity and association objects consist of a unique identifier (OID), a set of *attributes* (some OODB systems refer to them as *instance variables*) and a set of *methods*. Entity objects are real-world entities similar to those defined in the ER model. Association objects store constraint definition and references to its participating entity objects. They are proposed in our extended model to define explicit relationships involving multiple entity objects, especially for ternary and higher number of participating entity objects. Our extended model requires all participating objects of an association object to be entity objects.

A *primitive* attribute is an attribute that takes on system-defined data types such as integers, real numbers, and strings. Some OODB systems provide more powerful abstract data types such as *lists*, *arrays*, and *bags*. A *complex* attribute is defined over an existing user-defined object class. In the current OODB, complex entities are the only means by which the user can define real-world abstractions beyond generalization and specialization. A complex attribute can always be modeled as an external association object. In our extended model, we also allow *derived* attributes whose values are computed from attached inferencing rules expressed in Prolog-like expressions. We also require that association objects include only primitive and derived attributes.

A *primitive object* is an entity object whose attributes are primitive only. We define a *complex object* as an entity object that contains complex attributes, and a *composite object* as a complex object that contains explicit *PART_OF* references to other component objects in the definition. A composite object is always a complex object, but not vice versa.

Much confusion and trouble arise when database schema designers and

application developers define new abstractions using the reference link of complex entities in the current OODB model. The process typically requires repetitive and potentially conflicting interpretations that have to be implemented by end-user applications. Some OODBs define their own interpretations for selected abstractions. Such OODBs, however, lack generalized treatment in their interaction and interrelationship. It becomes difficult to add and integrate new abstractions into existing data models without retrofitting existing ones.

The reason for extending existing OODB models is to remedy such deficiency. We recognize that it is impossible for any database models to include formal interpretation on the infinite real-world semantic abstractions. In our extended model, we have taken a different approach to this issue. We studied and investigated various database semantic schemes and carefully selected a set of most frequently used abstractions so that we can formally define their semantics and their interrelationships. We also foresee a methodology to be developed for database designers to integrate new semantic abstractions into existing ones without revamping existing models. The formal extended model for PDKB is presented in Chapter 6.

Chapter 4

Survey of Related Work in Coupling Knowledge-Based Systems with Database Systems

The research prototype system we studied is coupling Proteus [36] with Orion [19] from Microelectronics and Computer Technology Corporation (MCC) [37]. Proteus is a frame-based nonmonotonic inference system, and Orion is an object-oriented database systems. Both systems are developed at the Advanced Computer Architecture Program in MCC. The commercial product we investigated is KEEconnection from IntelliCorp, a software product to bridge between knowledge-based applications built with IntelliCorp's Knowledge Engineering Environment system (KEE) and commercial SQL-based relational database systems.

4.1 Comparison with the Coupling of Proteus and Orion

We begin with a brief overview of both Proteus and Orion systems, and then describe their specific disadvantages when compared with our approach.

Proteus Overview. Proteus is a frame-based, nonmonotonic, truth maintenance system (TMS) that records logical inferences and dependencies among data. It allows efficient revision of a set of beliefs to accommodate new information, the retraction of a premise, or the discovery of a contradiction. It also facilitates the generation of coherent explanations. Each element of Proteus represents a potential belief with its status reflected in the *support-status* of the datum. This status is assigned by the TMS in accordance with a list of *justifications* attached to the datum. The Proteus TMS is *complete* because, with any set of justifications, it will achieve a stable well-founded state if such a state exists; otherwise it will recognize and report failure.

The data on which the TMS operates represent statements about objects, also called *frames*. The initial state of the system has several frames (one of these, *CLASS*, plays a special role as discussed below). Other frames are system defined or are built-in frames used as primitive objects. The user may enlarge this set by creating new frames, one at a time. Proteus supports the standard notion of *inheritance* as defined in the object-oriented programming paradigm. Proteus also supports the notion of *metaclass*, which serves as the basis for instantiating other classes of objects. Additional metaclasses may be created by linking any user-defined classes to the system *CLASS*.

Orion Overview. In Orion, all conceptual entities are modeled as objects. A primitive integer or string is as much an object as is a complex assembly of objects, such as an automobile. An object consists of some private memory that holds its state. The private memory consists of the values for a collection of attributes. The value of an attribute is itself an object, and therefore has its own private memory for its state (i.e., its attributes). A primitive object, such as an integer has no attributes. It has only a value, which is the object itself. More complex objects contain attributes through which they reference other objects, which in turn contain their own attributes.

The behavior of an object is encapsulated not only in attributes but also in *methods*. Methods consist of arbitrary codes that manipulate or return the state of an object. Objects can communicate with one another through messages, which constitute the public interface of the object. For each message understood by an object, there will be a corresponding method and a returned object. The messages include user-defined messages, access messages, and system-defined function   A user-defined message is a message

to a corresponding user-defined method already stored in Orion. An access message retrieves or updates the value of an attribute of a class. System-defined functions include all Orion functions for schema definition, creation and deletion of instances, transaction management, and so on.

Mapping the Proteus Model to the Orion Model. Difficulties arise when mapping two different systems that have been designed and implemented independently. In the case of mapping Proteus with Orion, the single most significant obstacle is Orion's lack of a metaclass concept that Proteus requires. Other than that, the two models share a high degree of basic similarity and are compatible in most respects. For example, the frame notion in Proteus can easily be mapped to the object definition in Orion. In general, the two models have the following elements in common:

- Instance relation: A class consists of a set of instances, and an instance may be a member of exactly one class.

- Attributes: A class consists of a set of attributes.

- Domains: An attribute may be typeless or typed. If a type is specified, the type is a class, possibly with its own set of attributes.

- Subclass relation on a class hierarchy.

- Message passing: Orion implements message passing as LISP function calling. This means that all Proteus interactions with Orion are implemented as LISP function calls to Orion, making it easier for Proteus developers to interface with Orion.

In the coupling approach taken by Proteus and Orion, Proteus is no different from other applications built on top of an Orion database system. That is, Proteus interactions to Orion are through functions calls, and Orion never calls Proteus. Orion simply provides a persistent storehouse for storing instances of Proteus frames which would otherwise be volatile. The database engine remains separate from the inference engine, and the knowledge system maintains the knowledge body within its own domain; i.e., the knowledge is separate from the data. Thus, Orion serves simply as a passive object storage awaiting Proteus for access. It is conceivable that other types of

knowledge systems, such as rule-based systems, could also be coupled to and run concurrently with Proteus on the same Orion databases.

The ramification of this coupling approach is twofold. First, since the knowledge system maintains its own knowledge body in its own application domain, the knowledge cannot be shared with other domains. For example, a manufacturing corporation would need to maintain one distinct knowledge system for corporate data, and another for manufacturing process control and management. Although both knowledge systems could be built on top of the same database system, there is no way for both bodies of knowledge to interact with each other (unless an interface application is implemented by knowledge engineers). Second, all inferencing tasks are performed by the knowledge system rather than by the databases. This means that the database cannot answer intelligent queries except by going through the knowledge system, and every knowledge system needs to provide its own querying mechanism to end users rather than having them use the querying mechanism provided by the databases. In essence, the coupling between Proteus and Orion represents one type of loose coupling strategy between knowledge systems and database systems with similar data models. .

4.2   Comparison with KEEconnection

The following begins with an overview of how to map from KEEconnection to relational database systems, and then describes KEEconnection's specific disadvantages when compared with our approach.

Mapping from KEEconnection to a Relational DBMS. The architecture of KEEconnection divides the process of building and operating a database-to-knowledge-base connection into two distinct tasks: First, KEEconnection, guided by input from the application developer, builds a *mapping knowledge base*, a description of how the data structures in a particular relational database should be related to data structures in a particular KEE knowledge base. KEEconnection then interprets the mapping knowledge base for the purpose of translating queries and transforming data. This mapping enables developers to connect a knowledge base to more than one database, and even to databases running on heterogeneous systems.

The three principal software modules of KEEconnection are mapping,

translation, and data communication. The greatest effort in this process is required in the mapping phase Once a mapping is completed, KEEconnection's translation and data communications modules take over, dynamically translating the applications's requests for data into SQL queries, managing network communications, and transforming downloaded data into the format specified by the mapping.

It is no surprise that the mapping phase is the single most important and time consuming process. In essence, the knowledge application developers (or knowledge engineers) need to define the corresponding relational database schema definition for those knowledge structures (or class *units*) used in knowledge systems.

These units correspond one to one with the database tables selected by developers. For each column in a table, KEEconnection creates a *slot* of the name in the table's corresponding KEE unit. These units will serve as templates for units that KEEconnection will later create to store downloaded data.

In addition to creating template units in the application knowledge base, KEEconnection also builds a separate mapping knowledge base. Units in the mapping knowledge base store information about how columns in database tables are mapped to slots in knowledge base units. Each template unit has its own mapping unit to maintain its database connections. Currently, KEEconnection supports access to relational database vendor products such as Oracle, Sybase, and Ingres.

Mapping Mismatch between KEE and Relational DBMS. Relational database tables consist of *rows* (tuples or records), which in turn are collections of *columns* (fields). The KEE knowledge base, on the other hand, is composed of collections data structures called *units*, which in turn use collections of *slots* to group together all relevant information about the entity, an object or concepts, represented by the unit. In this unit and slot, they support far more complex structures than their relational database counterparts. While a slot, like a database column, may be used to store primitive data such as numbers and strings, it may also store complex data types or complex objects such as arrays, bitmaps, hash tables, lists, and even references to other units.

Slots may also contain procedural information (programs or subroutines)

that will enable a unit to represent an object's behavior as well as its declarative characteristics such as size and shape. Such embedded procedures, called *methods*, are activated by a process called *message passing*. Special types of units called *active values* may be attached to a slot for the purpose of monitoring the slot's values and automatically performing a procedure when a new value is entered or the slot value is accessed. Active values are particularly useful for simulating alarm behavior or for a feedback system.

Knowledge system developers may also use logic-based *rule* language to represent knowledge about an object or a system's behavior, in which rules are also KEE units. The ability to store procedural information is only one of the ways in which knowledge base structures prove to be more expressive than database systems. In relational databases, each column in a table typically contains only one value, which often results in the break-up of logically-associated data into separate tables, or normalized tables. In KEE, a single slot often has multiple values. Moreover, the *frame* units provided by the structure not only allow logically associated data to be stored together but also allow explicit links between related data.

Thus the expressive data representation facilities provided by the KEE environment enable developers to capture in knowledge bases much of the semantic meaning lost in the relatively fragmented descriptions provided by the tables, columns, and rows of databases. Knowledge bases, with their interacting units composed of multivalued slots and facets, which are organized into hierarchies, support the richer and more detailed modeling of complex systems needed to automate knowledge-intensive tasks.

It is understandable that commercial knowledge systems like KEE will choose to couple with existing relational databases for marketing purposes. There are trends and sound justifications for those knowledge systems to loosely couple with some emerging OODB products in the future. But, by and large, there will always be some mismatches in terms of semantic modeling representation between them.

Chapter 5

Justification for the PDKB Coupling Strategy

Semantic networks in knowledge-based systems are very similar to the object-oriented database model. The inheritance property in OODB was originally borrowed from the frame concept in knowledge-based systems. The major difference is that semantic networks treat instances of object classes in OODB as objects rather than as instances. Semantic networks also allow arbitrary link types to be defined to connect various objects, as compared to the three fundamental links, generalization, composition, and aggregation, defined in the OODB model.

In the past, the AI and DBMS communities have aimed for different emphases in their research. The AI community has focused on inference and logical deduction to provide a more powerful reasoning engine based on semantic networks and frames. The OODB community has concentrated both on refining the OODB model to provide a more rigorous formalism and on other implementation considerations.

Recently the integration of knowledge-based systems with OODBs has gained tremendous interest and is particularly appealing to users who wish to manipulate the semantics of databases in a straightforward fashion, in contrast to approaches where semantics are "hard-wired" into database models. This removes the need to predefine complex semantic constructs within database host languages. The following capabilities represent a partial list of potential benefits:

- Sophisticated queries can be expressed more intelligently, such as transitive-closure queries and recursive queries.

- Intelligent front-ends such as natural language or voice recognition can be provided to increase user friendliness.

- Expert rules can be programmed with the database data in the same framework and environment.

- Database integrity constraints of databases can be expressed and enforced in a natural fashion.

- Large knowledge-based systems can use the concurrency and transaction control and other OODB features to make knowledge objects persist.

Most early connection schemes built between knowledge-based systems and relational DBMSs used loose-coupling techniques. Typically, they involved spawning SQL queries on the fly (dynamic queries), and passing data through the import/export mechanism of the two systems. This tends to result in poor performance. Besides degrading performance, the coupling of knowledge-based systems with relational DBMSs also presents difficulty in integrating two different data models under the same framework. In knowledge-based systems, graph and pointer traversals are fundamental operations. In relational DBMSs, associative access is known for its poor performance due to its value-based retrieval rather than pointer traversal when joining multiple tables to materialize end-user views.

Knowledge-based systems appears to couple better with OODBs than with relational DBMSs, at least from the perspective of data modeling. For example, with different retrieval schemes between relational DBMSs and knowledge-based systems (set-at-a-time versus tuple-at-a-time schemes), the inference engine requires higher access overhead without bypassing the multiuser concurrency control and recovery primitives. The transferring of an entire set of tuples of a retrieval request into the knowledge-based system workspace for reasoning is computationally expensive. The reason is that the inference engine typically needs to have only the first valid answer until it needs to backtrack, after which other answers may be sought.

Knowledge-based systems that are tightly coupled and closely integrated with database systems require new designs and implementations. Both systems must extend their interactions for rule tracing, value binding, and error checking. In a multiuser environment, concurrent access to one knowledge base must also be controlled. Most knowledge-based systems fail to address this issue. Access conflicts arise when trying to assert or retreat a fact or a rule used by another concurrent transaction. Recovery mechanisms for knowledge-based systems should also provide conventional database systems services such as undo logs and group commit operations.

The following are some of the potential issues, at different levels of interaction, to be addressed in tight coupling of these systems:

- At the language level, a logical syntax for the overall system and the underlying architecture that will allow coupling the database host language, the query languages, and the reasoning languages.

- At the program-development level, applications must be developed using both knowledge-base and database techniques.

- At the user environment level, the user needs to be able to interact with only one system instead of two.

- At the data and rules or knowledge representation level, a tight coupling of both data and knowledge semantics—using the same data model—is required.

- At the data and knowledge processing level, a tight coupling on both inference and database engines is required. For example, forward/backward inferencing or attribute value-binding in knowledge-based systems must be coupled with the query evaluation in database systems.

Based on current technologies in knowledge-base systems and database systems, we feel that the best result will be achieved by tight coupling of semantic-network-based knowledge systems with OODBs. The reasons are these:

- A fundamental requirement in coupling is to integrate a logic-based knowledge system with semantic databases both procedurally and semantically. However, the current research in semantic data models has

made little progress in formalizing the semantic representations, and is far from offering a practical implementation.

- Rule-based systems can also be represented using semantic networks, and frames and O-A-V representations can be categorized as subsets of semantic networks.

- Chapter 4 explained the drawbacks in coupling relational databases with knowledge-based systems from performance and data modeling perspective; these drawbacks are eliminated by using OODBs.

We feel that coupling semantic networks based knowledge systems with object-oriented databases offers the most natural integration from the perspective of both data modeling and knowledge representation. It also provides better performance with the currently known implementation techniques such as high-speed caching, indexing, clustering, which have been implemented by numerous OODB vendors.

Chapter 6

Formal Framework for PDKB

This chapter presents our formal model and representation of various semantic abstractions defined by PDKB. As defined previously, a semantic network is a graph whose nodes represent concepts, classes, or set of entities and arc models relationships obtained between these concepts. Besides the named relationships obtained from the modeling of an application with a semantic network, some special, structural relationships between concepts, called abstraction hierarchy, can occur and have a rich expressive power. Some of the most frequently used abstractions are *Generalization/Specialization* (also known as *IS_A* relationship), *Aggregation*, *Composition* (also known as *PART_OF* relationship), *Association*, and *Grouping* (also known as *ELE-MENT_OF*).

In a semantic network, knowledge engineers may define arbitrary named links that relate multiple entities (or object classes) with the specific semantics associated with them. Most, if not all, of these semantic-network-based knowledge systems however, interpret the semantics and the interrelationships of these abstractions in an ad hoc fashion; i.e., the systems extract and decode their semantics by traversing the net at individual basis.

For example, let us consider a simplified semantic-network knowledge base about manufacturing enterprises as shown in Figure 6.1, which encodes the following knowledge:

- A *Company* may manufacture multiple *Products*, and each product may be manufactured in different *Plants* owned by the company. This relationship is captured by the associate object *Product_Made* in Figure 6.1.
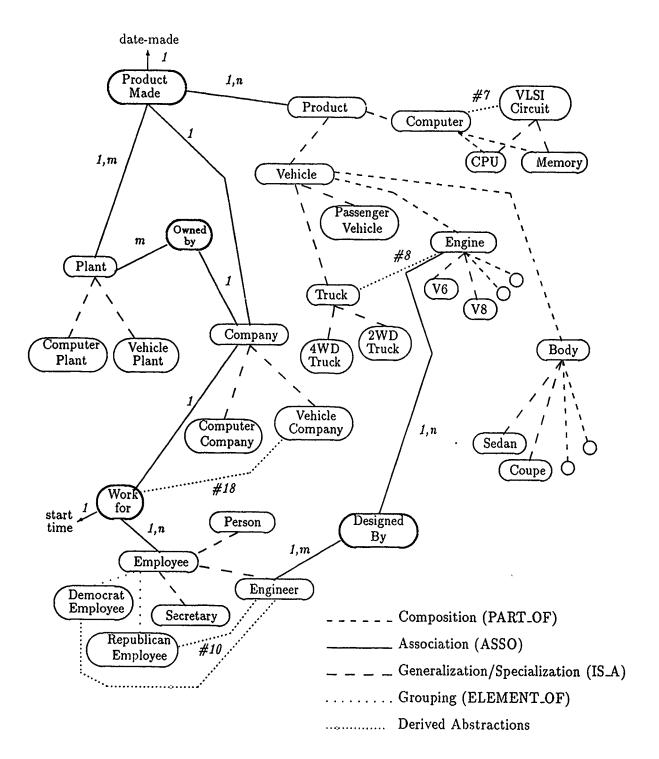
Figure 6.1: Semantic-Network Abstractions for Manufacturing Enterprises

Note that cardinalities between *Product_Made* and its participating entity objects are labeled with "1" for exact one, and "1,m" or "1,n" for one or many on their links.

- A *Plant* (entity object) can either be a *Computer Plant* or a *Vehicle Plant*, and a *Company* can either be a *Computer Company* or a *Vehicle Company*.

- A *Product* can either be a *Computer* or a *Vehicle*.

- A *Computer* is composed of *CPUs* and *Memories*, which are *VLSI Circuits*.

- A *Vehicle* can either be a *Passenger Car* or a *Truck*, and a *Truck* can be further specialized to be either a four-wheel drive (*4WD*) or a two-wheel drive (*2WD*). For any *Vehicles*, there must be an *Engine* and a *Body* as the first level-parts in their part hierarchy.

- An *Engine* can be a *V8* or *V6*, and the body style of a vehicle can be either *Sedan* or *Coupe*.

- A *Company* can have multiple *Employees* working for it, and any *Employee* can work for only one company at a given time.

- An *Employee* who is a *Person* can be a *Secretary* or an *Engineer*.

- *Employees* are grouped into two categories, either *Democrats* or *Republicans*.

- An *Engineer* may participate in many engine designs, and an *Engine* may be designed by multiple engineers.

Normally, a semantic-network based knowledge system will answer a question like "What are the parts that compose a *4WD Truck?*" by traversing the net starting from the *4WD Truck*. An intuitive algorithm to traverse the net is called the "shortest-path heuristic." This algorithm basically looks up all links simultaneously, and takes the value encountered on the shortest path from the node at which it starts. In this query example, it will visit the *Truck* entity (or class) and proceed to *vehicle* and further until it finds that the *vehicle* entity has encoded the knowledge about the general parts

hierarchy for a *Vehicle*. It then combines the hierarchy with the specific parts that are unique about the 4WD design. As seen from the description, this algorithm shows two shortcomings: first, it has to traverse every node and interpret the node's semantics and interrelationships based on the types of links (or abstractions) it visits. Second, Brachman and Levesque [43] along with Reiter and Criscuolo [44] pointed out, how such a strictly syntactic symbolic approach can go wrong when they are devised without worrying about what the representation is about.

To remedy these shortcomings, we propose a set of formalized axioms [1] that will provide automatic inferencing for system-defined abstractions. For example, a specialized entity such as *4WD* should inherit the generic parts-hierarchy information associated with its superclasses without having to traverse the net explicitly. It is also obvious that, for any two abstractions that have been defined in the semantic network, there should also exist well-defined formal semantics on how they should interact with each other, rather than leaving them to be interpreted in an arbitrary and ad hoc fashion. This leads us to define the following set of axioms to describe the formal semantics of those abstractions.

6.1   Formalized PDKB System-Defined Inferencing Axioms

1. $IS\_A(X,Y) \& IS\_A(Y,X) \Longleftrightarrow X = Y$

2. $IS\_A(X,Y) \& IS\_A(Y,Z) \Longrightarrow IS\_A(X,Z)$

3. $IS\_A(X,Y) \& IS\_A(X,Z) \Longrightarrow \exists W (IS\_A(Y,W) \& IS\_A(Z,W))$

4. $PART\_OF(X,Y) \Longrightarrow \neg PART\_OF(Y,X)$

5. $PART\_OF(X,Y) \& PART\_OF(Y,Z) \Longrightarrow PART\_OF(X,Z)$

6. $IS\_A(X,Y)\&PART\_OF(Y,Z) \Longrightarrow PART\_OF(X,Z)$
   and
   $PART\_OF(X,Y)\&IS\_A(Y,Z) \Longrightarrow PART\_OF(X,Z)$
   are not necessarily true.

7. $IS\_A(X,Z)\&PART\_OF(X,Y) \Longrightarrow PART\_OF(Z,Y)$

8. $IS\_A(Y,Z)\&PART\_OF(X,Z) \Longrightarrow PART\_OF(X,Y)$

9. $IS\_A(X,Y)\&ELEMENT\_OF(Y,Z) \Longrightarrow ELEMENT\_OF(X,Z)$

10. $IS\_A(Y,Z)\&ELEMENT\_OF(X,Z) \Longrightarrow ELEMENT\_OF(X,Y)$

11. $INSTANCE\_OF(x,X)\&IS\_A(X,Y) \Longrightarrow INSTANCE\_OF(x,Y)$

12. $IS\_PART\_OF(x,y)\&INSTANCE\_OF(x,X)\&INSTANCE\_OF(y,Y)$
    $\Longrightarrow PART\_OF(X,Y)$

13. $PART\_OF(X,Y)\&INSTANCE\_OF(y,Y)$
    $\Longrightarrow \exists x(INSTANCE\_OF(x,X)\&IS\_PART\_OF(x,y))$

14. $IS\_ELEMENT\_OF(x,y)\&INSTANCE\_OF(x,X)\&INSTANCE\_OF(y,Y)$
    $\Longrightarrow ELEMENT\_OF(X,Y)$

15. $INSTANCE\_OF(x.X)\&ELEMENT\_OF(Y,X)$
    $\Longrightarrow (\forall z(IS\_ELEMENT\_OF(z,x) \Longrightarrow INSTANCE\_OF(z,Y))$

16. $ASSO(X,Y) \Longrightarrow ASSO(Y,X)$

17. $ASSO(X,Y)\&ASSO(Y,Z) \Longrightarrow ASSO(X,Z)$

18. $ASSO(X,Y)\&\exists Z(IS\_A(Z,Y)) \Longrightarrow \exists W(IS\_A(W,Y)\&ASSO(X,W))$

19. For each P where P is a structural and behavioral property of Z, we have

    $IS\_A(X.Y)\&\exists Z(PART\_OF(Z,Y)) \Longrightarrow (P(Z) \Longrightarrow P(X)).$

Axiom 1 states that if object class X is a Y and Y is an X, then X must be the same as Y. Axiom 2 defines the transitive property for *IS_A* abstraction. Axiom 3 states that if there exist multiple inheritances Y and Z for object class X, then there must also exist a common superclass, W, for both Y and Z. This property guarantees that all object classes must be ultimately rooted from a system defined class. Axiom 4 disallows cycles in the composition hierarchy. Axiom 5 states the transitive property in the composition hierarchy. Axiom 6 states that transitive closure is not always true in mixing both composition and generalization abstractions. For example, in Figure 6.1, a V6-engine is not necessarily part of a vehicle (the vehicle could have a V8-engine), and an engine is not necessarily part of a product (the product could be a computer).

Axiom 7 states that if object class X is a part of Z, then all superclasses of X are also part of Z. This derived abstraction is illustrated in Figure 6.1, with the link labeled "#7" between computer and VLSI circuit, i.e., if a CPU is a part of a computer, then VLSI circuits are also a part of the computer. Axiom 8 states that an object class Y inherits the part-hierarchy property of its superclass, Z. The link labeled "#8" in Figure 6.1 illustrated this property; i.e., if an engine is a part of a vehicle, it is also a part of a truck. Axioms 9 and 10 illustrate the relationship between the *IS_A* and *ELEMENT_OF* abstractions. For example, the link labeled "#10" illustrates that if a Democrat employee is an element of object-class employee, it is also an element of engineer. Axioms 12 to 15 capture various properties between *PART_OF*, *IS_PART_OF*, *INSTANCE_OF*, and *IS_INSTANCE_OF* abstractions.

From Axiom 16 to 20, an *ASSO* abstraction [39] [40] [41] captures arbitrary relationships among object classes. The *ASSO* abstraction takes two parameters of generic object class as their domains. Axiom 18 states that if object class X is associated with Y and there exists a subclass Z of Y, then X is also associated with one of Y's subclasses, W, where W could be Z. For example, the link labeled "#18" in Figure 6.1 illustrates that an employee must work for either a vehicle company or computer company, which are both subclasses of the class company. Axiom 19 is an axiom scheme which states that a subclass X of Y inherits all *structural* and *behavioral*[1] properties

---

[1]A structurally object-oriented model supports complexly structured objects, i.e., ob-

[38] from Y's part hierarchy. For example, a 4WD truck should possess all properties of an engine which is a part of vehicle, such as horse power rating (a behavioral property) and cylinder and valve part components (structural property). Note that we define P (property) using second order logic as a function of object classes X and Z.

6.2   User Defined Inferencing Rules

The formally defined PDKB system-defined abstractions, using Prolog-like expressions, are only part of the picture. In this section, we describe how to define user and application-specific inferencing rules for both associate and entity objects. There are two types of user-defined rules: *deductive rules* and *constraints*. A user-defined deductive rule can be used for two different purposes: to compute the value for a derived attribute, or to supply other deductive information. In the first case, the value of a derived attribute is computed from the (derived) attributes of other entity objects or of itself. In the second case, a rule simultaneously instantiates several attributes (an extension of rules for derived attributes).

A user-defined deductive rule consists of a *head h* and a *body p_1&...&p_n* of the form p_1&...&p_n $\Longrightarrow$ h, where each p_i is called a *subgoal*. The predicate name, which is the head of a rule, is the same as the derived attribute the rule describes. The body of a rule describes how to compute the value for a derived attribute. It consists of predicates, comparison operations, and simple arithmetic operations. Constraints are user-defined rules without heads, i.e., they do not compute the value of a variable, but rather state a base fact. A rule can reference attributes and derive attributes within the same object and (derived) attributes defined along the inheritance hierarchy. Besides attributes and rules, predicates can also match with other (derived) attributes anywhere from the system schema. However, the PDKB requires that objects that own these (derived) attributes be in some way connected to the object along the search path within the network.

In addition to system-defined axioms and user-defined rules, PDKB au-

---

jects that may be constructed from subobjects (which are objects in their own right) and thus go beyond incorporating simple primitive attributes A behavioral object-oriented model provides facilities that allow the definition of new object types, including the appropriate type-specific operators.

tomatically generates Prolog-like rules for certain schema definitions, such as unique attributes and cardinalities among participating entity objects for association objects. This allows complete semantic information to be encoded in logic rather than arbitrary code implementation in PDKB. We shall illustrate this concept with a running example concerning the knowledge about entity object classes *Employee*, *Company*, and *Person* as shown in Figure 6.1. Throughout this example, we use C++-like class declarations to define them. We recommend readers to focus on the concepts and techniques illustrated in the example, rather than the specific language syntax and constructs, which may change as we continue to develop new notations to improve their clarity and expressiveness.

```
ENTITY Person
    ATTRIBUTES
        Name: STRING;
        UNIQUE Social_Sec_Num: INTEGER;
        Home_Address: Address;
        Birth_Day: Date;
        Gender: (male, female);
END Person
```

The UNIQUE qualifier on Social_Sec_Num attribute within the *Person* object, which ensures that no two-person instances have the same social security number, automatically generates the following rule when it is declared in the PDKB:

$$Social\_Sec\_Num(P_1, S) \& Social\_Sec\_Num(P_2, S) \implies P_1 = P_2$$

```
ENTITY Employee IS_A Person
    ATTRIBUTES
        Salary: REAL;
        Office_Phone_Num: INTEGER;
    RULES
```

$$Work\_For(Emp, Comp, Time) \& Owner(Comp, P) \implies Boss\_Of(Emp, P);$$

```
END Employee
```

The above user-defined rule for entity object *Employee*, defines a derived attribute *Boss_Of* to state the fact that the owner of a company is also the boss of its employees. *Work_For* is an association object defined below, and *Owner* is an attribute defined in the entity object *Company*. PDKB generates a logic rule $IS\_A(Employee, Person)$ on the IS_A abstraction.

```
ASSOCIATION Work_For
    BETWEEN Employee, UNIQUE Company
    ATTRIBUTE
      UNIQUE Starting_Time: Date;
    RULES
      Starting_Time >= Founding_Time;
END Work_For
```

The rule in the above association object *Work_For* defines the constraint that no employee can start working for a company unless the company has been founded. Five extra rules are generated automatically from the above declaration. The first two rules enforce a unique date for an employee to start working for a company. The remaining rules define generic binary association relationships among participating entities:

$$Work\_For(Emp, Comp_1, Time) \& Work\_For(Emp, Comp_2, Time)$$
$$\implies Comp_1 = Comp_2$$

$$Work\_For(Emp, Comp, Time_1) \& Work\_For(Emp, Comp, Time_2)$$
$$\implies Time_1 = Time_2$$

$$Work\_For(Emp, Comp, Time) \implies ASSO(Emp, Comp)$$

$$Work\_For(Emp, Comp, Time) \implies ASSO(Emp, Time)$$

$$Work\_For(Emp, Comp, Time) \implies ASSO(Comp, Time)$$

The following is the definition for object class *Company*:

```
ENTITY Company
    ATTRIBUTES
      C_Name: STRING,
```

```
        Location: Address;
        Revenue: REAL;
        Owner: Person;
        Founding_Time: Date;
END Company
```

Following is the definition for association object *Product_Made*. The constraint in the rule section defines that the manufacturer must own the plant if its products are to be manufactured from that plant. We also assume that the *Owns* association object has been defined elsewhere.

```
ASSOCIATION Product_Made
    BETWEEN Plant, UNIQUE Company, Product
    ATTRIBUTE UNIQUE Date_Made: Date;
    RULES

        Owns(Comp, Plant)

END Product_Made
```

Five rules are generated from the above association object *Product_Made* declaration. The first two enforce their cardinalities, and the last three define their generic binary association relationships:

$$Product\_Made(Plant, Comp, Prod, Date_1) \& Product\_Made(Plant, Comp, Prod, Date_2)$$
$$\implies Date_1 = Date_2$$

$$Product\_Made(Plant, Comp_1, Prod, Date) \& Product\_Made(Plant, Comp_2, Prod, Date)$$
$$\implies Comp_1 = Comp_2$$

$$Product\_Made(Plant, Comp, Prod, Date) \implies ASSO(Plant, Comp)$$

$$Product\_Made(Plant, Comp, Prod, Date) \implies ASSO(Comp, Prod)$$

$$Product\_Made(Plant, Comp, Prod, Date) \implies ASSO(Comp, Date)$$

The following are definitions for entity objects *Product*, *Vehicle*, and *Engine*:

```
ENTITY Product
   ATTRIBUTES
      UNIQUE Product_ID: INTEGER;
      Product_Name: STRING;
END Product

ENTITY Vehicle IS_A Product
   ATTRIBUTE
      UNIQUE Vehicle_ID: STRING;
      Model: STRING;
   PARTS
      Engine:1;
      Body:1;
END Vehicle
```

Six rules are generated from the PARTS declaration in the *Vehicle* entity object definition. The first two rules define the composition (or *PART_OF*) hierarchy for vehicles. The third and fourth rules define that there is exactly one engine and one set of body components for each vehicle. The fifth rule defines the uniqueness of a vehicle identifier. The last rule defines the logic for the IS_A abstraction:

$$PART\_OF(Engine, Vehicle)$$

$$PART\_OF(Body, Vehicle)$$

$$PART\_OF(Engine_1, Vehicle) \& PART\_OF(Engine_2, Vehicle)$$
$$\implies Engine_1 = Engine_2$$

$$PART\_OF(Body_1, Vehicle) \& PART\_OF(body_2, Vehicle)$$
$$\implies Body_1 = Body_2$$

$$Vehicle\_ID(Vehicle_1, ID) \& Vehicle\_ID(Vehicle_2, ID)$$
$$\implies Vehicle_1 = Vehicle_2$$

$$IS\_A(Vehicle, Product)$$

```
ENTITY Truck IS_A Vehicle
```

```
    ATTRIBUTES
      Payload: REAL;
    RULES
      Payload > 4000 /*lbs */
END Product

ENTITY 4WD_Truck IS_A Truck
    ATTRIBUTES
      4WD_Differential_Ratio: REAL;
    PARTS
      4WD_Differential;
END 4WD_Truck
```

The following rules are generated for the parts hierarchy that is specific to a 4WD truck, and the IS_A abstraction:

$$PART\_OF(4WD\_Differential, 4WD\_Truck)$$

$$IS\_A(Truck, Vehicle)$$

$$IS\_A(4WD\_Truck, Truck)$$

From the above definition, PDKB automatically infers from system-defined Axiom 8 that all trucks have an engine and body as their first-level components in their parts hierarchy. The rule in *4WD_Truck* also defines an extra part *4WD_Differential* other than the engine and body parts it inferred from *4WD_Truck*.

Chapter 7

Query and Inference Processing in PDKB

Search is the fundamental process in both knowledge-base systems and databases. In knowledge-base systems, searching supports inferencing; in databases, searching supports query evaluation. However, the two systems search in different ways. In this chapter, we compare the differences to draw conclusions about how the different search modes can be combined in the PDKB system so that we can develop a strategy for query and inferencing processing.

In deductive inferencing, knowledge is represented in logic as a set of *formulae* (axioms or rules). A query is another formula with free variables to be bound to the logic at run time. Hence, the result of a query is the set of free-variable instantiations that are provable from the underlying logic.

Search in query evaluation is much simpler than search in deductive inferencing. Only the interpretation needs to be searched as opposed to an arbitrarily large number of rules. The knowledge representation for deductive inference is more powerful than the knowledge representation for query evaluation. Specifically, there are sets of axioms that cannot be captured by the combination of an interpretation and a set of definitions. Also, the deductive inferencing approach is more flexible than the query evaluation approach. Specifically, deductive inferencing allows any knowledge to be added to the representation without affecting knowledge already represented.

In most applications, some knowledge will never change. It is best to build this knowledge into the search process as an inherent mechanism, so that the

query evaluation and inferencing performance can benefit from the built-in optimization strategies. In deductive inferencing, integrity constraints are not distinguished from other knowledge and the search process cannot take advantage of them. In many applications, integrity constraints are used for detecting and correcting errors in input data. In this situation, constraint knowledge must be distinguished from other knowledge. The effect of moving from a knowledge representation for deductive inference to one for query evaluation is to transform the axioms to a simpler form. A more efficient deductive search is possible if the axioms and rules are restricted to conjunctive forms of simple predicate clauses, i.e., no negations, disjunctions, or existing quantifiers. Queries, however, are transformed into a more complex form, so that the overall effect is to transfer complexity from the axioms to the queries.

The fundamental difference between knowledge-base systems and database systems in processing queries is that most knowledge-base systems employ a very powerful, yet computationally expensive search mechanism, while databases employ a far less powerful, yet computationally economical search mechanism. The main reason for the difference is that database queries normally assume defined syntax with restricted semantics, such as SQL. This syntax allows the database query processor to build-in various optimization plans. Because it is bound dynamically by the run-time environment and the organization of logic rules, inference processing cannot be made less computation intensive.

To reach a balance between the efficiency of database query processing and the power of knowledge inferencing, we plan to provide PDKB with the following types of query and inference processing capabilities:

- The user will be able to ask the actual value of (derived) attributes of entities. This means that rules and constraints will be used to compute values, so that traditional database query processor can be employed to search for values.

- The user will be able to ask how an answer to a query is obtained. This means that relevant rules and constraints will be played back to the users.

- The user will be able to ask if a given hypothesis can be proved from the current knowledge base, and *backward chaining* will be triggered to

process the user's queries.

- The user will be able to ask the system to deduce new facts from existing knowledge, and *forward chaining* will be triggered.

A detailed query and inferencing processing design is beyond the scope of this research and remains for future work.

Chapter 8

Future Research Topics

We have presented and described a flexible model that will tightly couple semantic-network-based knowledge systems with OODB models. Some follow-up work remains to be done and there are several research topics that should be pursued in the future.

- Higher-Order Object-Oriented Logic Languages: Many higher-order logic languages, such as LDL [48], have been proposed to increase modeling and programming capabilities. Some higher-order languages such as HiLog [45] [46] were designed specifically for database inferencing. The development of LDL shows the difference between a set value and an abstract object whose state is a set. In our formal PDKB framework, we introduce a system-defined axiom, Axiom 19, which defines the behavioral and structural properties as second-order logic to the object. In our future research, we plan to extend the first-order logic in I DKB so that other objects with "higher orderness" such as metaschemata, objects properties, and functions can be modeled and treated as values. We expect to adopt and integrate relevant existing research into the PDKB model.

- External Interface Requirements: The current PDKB is designed as a standalone system and has no direct interfaces with other systems. We plan to provide PDKB with a generic interface to other external relational and OODB systems in a loosely coupled fashion. The detailed definition is beyond the scope of this project and remains for future work.

- Natural Language Processor: In current natural-language processing technology, there are two extreme categories. One extreme uses *key word* matching, which affords great flexibility but does not provide a precise understanding of the text. The other extreme uses *syntactic parsing*, which reflects the exact structure of the input sentences, but allows no flexibility. In the future PDKB natural-language-processing extension, we suggest adopting the notion of *semantic grammar*, proposed by Burton [11] [12]. This model offers a desirable middle path for parsing natural-language input. To achieve greater conciseness of definition and ease of development in cases involving more complex input sentences, the grammar implemented with proceduralized rules can be replaced by the formalism of augmented transition networks (*semantic ATNs*), compiled for better efficiency [13].

  The basic idea of a semantic grammar is to have rules decompose global categories into lower-level constituents as in a regular syntactic grammar. But whereas a regular grammar decomposes syntactic categories — say, a sentence into noun and verb phrases — a semantic grammar decomposes semantic categories, that is, domain concepts, into their constituents. The semantic decomposition is recursively repeated down to the different ways elementary concepts can be expressed in English. For instance, a given quantity will eventually become a number with the specification of a unit.

  We believe that a natural-language processor could provide end users with an intuitive interface to interact with the PDKB system.

- Machine Learning: The term *machine learning* has been used in a variety of AI applications. It generally refers to the ability of a program to discover or learn information by itself. A variety of programs that aim to learn have been developed and proposed. However, very few have addressed issues involved in self-learning database systems. It is desirable that an intelligent database system be able to extract and formulate knowledge from the underlying database automatically. One of the early ideas for programs that discover knowledge from data was the CLS algorithm developed by Hunt et al. [8]. In this system, a decision tree was generated by repeatedly separating a set of examples into smaller subsets, characterized by the values of attributes for each

example. However, this algorithm can easily go astray and, if not properly guided, will generate monstrous decision trees that make little sense.

More recently, the RX and Radix projects [9] [10] are pioneering works in distilling information from large databases by using statistical techniques. The technique used by those two projects determines the correlation between time-oriented events recorded in the database and uses that correlation to extract knowledge. However, because of the tremendous difficulty of the problem, this technology is still at its most rudimentary stages. The goal of machine learning is to write programs and eventually to build entire computer systems that can learn by themselves, as humans do.

The main difference between machine learning and its human counterpart ought to be one of speed. If a machine that learns at a speed considerably faster than humans cannot be built, learning machines will remain a curiosity. We feel that machine learning is the key to success for future integration of knowledge-base systems and database systems as the information and knowledge base grow exponentially.

For the PDKB, we would like to apply current existing techniques for machine learning, and also explore new theories in machine learning in the future.

- Off-the-Shelf Object-Oriented Database Systems: The current PDKB prototype implements a customized database for its own use. In the future, we would like to port it to commercial OODB systems running in UNIX-based heterogeneous networking environments. The reason for moving to commercial OODB systems is that it has become cost inefficient for an individual software system to develop its own database and endow it with functions equivalent to those of full-fledged OODB systems.

  However, coupling knowledge-based systems with existing commercial OODB systems requires the knowledge-base and database systems to be compatible from both data modeling and physical implementation perspectives. With this requirement in mind, we investigated various research prototypes and commercial OODBs, and categorized them into the following approaches:

- Extended Relational Database Systems: These systems are based on extensions of a relational query language by the addition of capabilities such as procedure attachment and abstract data type. Relational database systems provide simple and well-defined query language, for example, SQL, with sound, proven, mathematical representations such as relational algebra and tuple relational calculus. Researchers have proposed various extensions to remedy some well-known relational data model deficiencies, such as the lack of recursive query capability and support for complex data objects. Among the extended systems, the most powerful and representative one is probably POSTGRES [20], from the University of California at Berkeley. POSTGRES is so named because it is a follow-on to its predecessor INGRES, a fully implemented commercial relational database system from Relational Technology Incorporated. The POSTGRES system provides objects, Object IDentifiers (OIDs), compound objects, multiple inheritance, versions, historical data, procedures, and a powerful extended relational query language, POSTQUEL, which is named after INGRES's QUEL.

- Extended Functional Database Systems: These systems are based on the functional data model that uses a data access language based on mathematical function notations with a declarative functional query language to define function values. In this model, the query language is also extended to allow procedural specification of functions as well, thus becoming computationally complete. The basis of the functional model is *objects* and *functions*. Like other database models, primitive objects such as integers and strings are generally distinguished from entity objects, and the latter are represented by some notions of OID. Two research prototype systems belong to this approach: Iris from Hewlett-Packard [22] and PROBES [21] from Computer Corporation of America.

- Semi Object-Oriented Database Systems: This type of system defines an object-orientation abstraction on top of a relational model; that is, objects are defined as relational views that are further defined by a set of base tables. When a user accesses objects, views materialize from base tables. Basically, this approach

allows object semantics to be defined and maintained at the user's level. This is in contrast to the loss of semantics when the tables of relational tables are normalized. One research prototype belongs to this approach: Penguin [23] from the Computer Science Department of Stanford University. A similar approach was also proposed by Kung [24].

Note that although the Iris object-oriented database system from Hewlett-Packard belongs to the extended functional database systems approach, its implementation is based on an internal relational storage manager called DBCORE, the kernel of HPSQL, a HP relational database system. Objects in Iris are physically stored as a set of base tables in first normal form. When objects are retrieved, joins are used to materialize those objects.

– Full Object-Oriented Database Systems: A full OODB system can be built from the ground up using a strict object-oriented data model rather than imposing an object-oriented abstraction on top of an existing relational model. Currently, a host of commercial products and research prototypes use this full OODB approach. For research prototypes, Orion [18] from MCC has been regarded as the most representative work. Recently, Itasca Systems, Inc., began production of ORION. A number of commercial products are already on the market: Gemstone from Servio Logic, Ontos from Ontologic, Versant from Versant Object Technology, and ObjectStore from Object Design.

– Database System Generators: A database generator allows the user to build an OODB tailored to particular needs—for example, a customized data model and database host language. Typically, a user must have substantial internal database implementation knowledge to perform the customization. For example, the user may want to choose a specific concurrency control and transaction model that will work with a particular query optimization scheme for his own purpose. The database generator has basic building blocks with different options that users may configure. Two research prototypes belong to this approach: EXUDOS [25] and GENESIS [26].

- Object Managers: These systems generally provide fewer functions and features than systems in the other categories. In some cases, they are used as the storage management level of one of the more complete systems. Typically, they provide a persistent object store with some concurrency control, and they generally do not provide a query and programming language. They may be thought of as extensions of virtual memories for operating systems, as opposed to extensions of programming languages or database systems. Systems like MNEME [27] and POMS [28] belong to this approach.

After studying and analyzing these different object-oriented database systems, we find the database-system-generator approach most suitable for tight coupling of knowledge-based systems, since its basic building blocks are the most flexible for other systems to couple with. Its capability of allowing a customizable database model also provides tremendous efficiency for us to build the semantic networks into the system. However, none of those database system generators are commercially available. An alternative is to use full OODB products such as Gemstone, Ontos, and Itasca. We need to explore these systems in further detail, but such exploration is beyond the scope of this project.

- Real-Time, Memory-Resident Object-Oriented Database: In the past, some efforts have been made to implement memory-resident relational database systems, such as the Real-Time Database from Hewlett-Packard Industrial Application Center at Sunnyvale [47], to support real-time applications such as robotics. Although most OODB products on the market provide adequate performance by caching frequently used objects from disk, they are still not capable of the real-time performance required by time-critical applications. For the PDKB system, it would be advantageous to have its OODB be memory resident for optimum performance.

- Discretionary and Multilevel System Security: The current PDKB does not address multilevel security requirements. Although multilevel security is beyond the scope of this project, we believe that the future usability of the PDKB system would benefit from the development of an implementation specification for discretionary security control. A

companion study could be performed on how to apply multilevel security to the design of the prototype. Such an investigation would draw on the results of a joint study recently completed by SRI's Computer Science Laboratory and Artificial Intelligence Center to develop a security model for knowledge-based systems [49].

- Characterization and Integration of other Semantic Abstractions into PDKB: In real-world applications, various semantic abstractions need to be defined and characterized other than those that have been defined in the PDKB. For example, object versioning and configuration are two most commonly used abstractions for CAD/CAM and CASE applications. We could define formal logic and rules similar to the aggregate and composition abstractions into the PDKB framework so that these abstractions could interact with other abstractions in a seamless and consistent fashion. This would also require the development of methodologies and theories for adding new abstractions that could provide full interaction semantics without generating conflict with existing axioms and rules.

- Database queries and inference rules processing have been discussed in chapter 7.

Chapter 9

Conclusion

This report presents a flexible and powerful model for tight coupling of semantic-based knowledge systems with OODB systems. In the PDKB model, we propose a set of formalized axioms that define the semantics and their relationships for *generalization/specialization, aggregation, composition, association* and *grouping* abstractions. We also show how to define user specific inferencing rules and constraints in the semantic-network schema, and how to interact with system-defined axioms to process user queries and inferencing.

We believe that the PDKB model we have developed is the first to provide a complete formalized framework for both system-defined axioms and user-defined inferencing rules or constraints to model the knowledge representation of the real-world abstractions. We also believe that PDKB is the first attempt to tightly couple semantic-network-based knowledge systems with OODB models in a unified framework.

Bibliography

[1] U. Schiel, 1989: Abstractions in Semantic Networks: Axiom Schemata for Generalization, Aggregation and Grouping. *SIGART newsletter*, Number 107 (January)

[2] J.D. Ullman, 1988: *Principles of Databases and Knowledge-Based Systems: Vol. 1* Computer Science Press, Maryland

[3] M. Hammer and D. Mcleod, 1981: Database Description with SDM: A Semantic Database Model. *ACM Transactions on Database System*, Vol. 6, No. 3, pp 351 - 386. (September).

[4] J. Peckham and F. Maryanski, 1988: Semantic Data Models. *ACM Computing Surveys*, Vol. 20, No. 3, pp 143 - 189. (September).

[5] R. Hull and R. King, 1987: Semantic Database Modeling: Survey, Applications, and Research Issues. *ACM Computing Surveys*, Vol. 19, No. 3, pp 201 - 260. (September).

[6] T.J. Teorey, D. Yang and J.P. Fry, 1986: A Logical Design Methodology for Relational Databases Using the Extended Entity-Relationship Model. *ACM Computing Surveys*, Vol. 18, No. 2, pp 197 - 222. (June).

[7] Paul Harmon and David King, 1985: *Expert Systems – Artificial Intelligence in Business*. New York: John Wiley & Sons, Inc., p. 35.

[8] E. Hunt, et al, 1966: *Experiments in Induction*. New York: Academic Press

[9] R. Blum, 1982: Discovery, confirmation and incorporation of casual relationships from a large time oriented clinical database: The RX project. Computers and Biomedical Research.

[10] R. Blum, 1986: Computer assisted design of studies using routine clinical data: Analyzing the association of prednisone and serum cholesterol. *Annuals of Internal Medicine.*

[11] R.R. Burton, 1975: Semantically-Centered Parsing. Doctoral Dissertation, University of California, Irvine, CA.

[12] R.R. Burton, 1976: Semantic Grammars: An Engineering Technique for Constructing Natural Language Understanding Systems. BBN Report 3453 (ICAI) report 3. Bolt Beranek and Newman Inc., Cambridge, MA.

[13] R.R. Burton and J.S Brown, 1979: Toward a Natural Language Capability for Computer-Assisted Instruction. In O'Neil, H. (Ed.) *Procedures for Instructional Systems Development.* New York: Academic Press.

[14] D. C. Tsichritzis and F. H. Lochovsky, 1982: *Data Models.* Englewood Cliffs, NJ: Prentice-Hall, Inc. p 8.

[15] T. J. Teorey, D. Yang, and J. P. Fry, 1986: A Logical Design Methodology for Relational Databases Using the Extended Entity-Relationship Model. *ACM Computing Surveys*, Vol. 18, No. 2, pp 197 - 222, (June).

[16] P. Chen, 1976: The Entity-relationship Model – Toward a Unified View of Data. *ACM Transactions on Databases System*, Vol. 1, pp 9-36, (March).

[17] J. M. Smith and D.C.P. Smith, 1986: Database Abstractions: Aggregation and Generalization. *ACM Transactions on Database Systems*, Vol. 2, No. 2, June. pp 105 - 133.

[18] W. Kim, N. Ballou, H.-T. Chou, J.F. Garza, D. Woelk, 1989: Features of the Orion Object-Oriented Database System. In W. Kim and F.H. Lochovsky (Eds.), Object-Oriented Concepts, Database, and Applications, Reading, MA: Addison-Wesly, pp 262.

[19] W. Kim, 1990: Architectural Issues in Orion Object-Oriented Database. *Journal of Object-Oriented Programming*, Vol. 2, No. 6, pp 30, (Mar/April).

[20] R. Stonebraker and L. Rowe, 1986: The Design of POSTGRES. *Proceedings of SIGMOD Conference*, Washington DC.

[21] U. Dayal and J. Smith, 1985: PROBES: A Knowledge-Oriented Database Management System. *Proceedings of the Islamorada Workshop on Large Scale Knowledge Base and Reasoning Systems*, (February).

[22] D. Fishman et al, 1987: Iris: An Object-Oriented Database Management System. *ACM Trans. on Office Information System*, Vol. 5, (January).

[23] T. Barsalou, 1990: View Objects for Relational databases. Ph.D. Dissertation, Report No. STAN-CS-90-1310, Department of Computer Science and Medicine, Stanford University, Stanford, CA.

[24] C. Kung, 1990: Object Subclass Hierarchy in SQL: A Simple Approach. *Communications of the ACM*, Vol. 33, No. 7, pp. 117-125, (July).

[25] M. Carey, et al, 1986: The Architecture of the EXODUS Extensible DBMS. *Proc. of the Internat. Conf. on Object-Oriented Database Systems*, Asilomar, CA.

[26] D. Batory, 1986: GENESIS: A Project to Develop an Extensible Database Management System. *Proc. of the Internat. Workshop on Object-Oriented Systems*, Computer Society Press.

[27] J.E.B. Moss and S. Sinofsky, 1986: Managing Persistent Data with Mneme: Designing a Reliable Shared Object interface. *Proc. Internat. Workshop on Object-Oriented Database Systems*, Asilomar, CA.

[28] W. Cockshot, M. Atkinson and K. Chisholm, 1984: Persistent Object Management Systems. *Software – Practices and Experience*, vol. 14, 1, (January).

[29] D. Warren, L. Pereira, and F. Pereira, Prolog – the Language and its Implementation Compared with LISP. *Proc., Symposium on AI and Programming Languages*, SIGPLAN Notices, 12, No. 8.

[30] H. Gallaire, J. Minker, and J.-M. Nicolas, 1984: Logic and Databases, A Deductive Approach. *Computing Surveys*, Vol. 16, No. 2, pp. 153 - 185, (June).

[31] Rieter, 1984: Towards a Logical Reconstruction of Relational Database Theory. In M. L. Brodie, J. Mylopoulos and J. W. Schmidt (Eds.) *On Conceptual Modeling*, Berlin: Springer-Verlag.

[32] J. Almarode and T. L. Anderson, 1990: EasyObjects Designer: A Tool for Object-Oriented Database Design. Technical Report, Servio Logic Corporation, Beaverton, Oregon

[33] M. L. Brodie, 1984: On the Development of Data Models. In M. L. Brodie, J. Mylopoulos and J. W. Schmidt (Eds) *On Conceptual Modeling*, Berlin: Springer-Verlag.

[34] E.-C. Chow, 1987: Representing Databases in Frames *Proc. of AAAI-87*, pp. 405-410, Seattle, WA: Amer. Assoc. for Artificial Intelligence.

[35] J. Moad, 1987: Building a Bridge to Expert Systems. *Datamation*, pp. 17-19, (Jan).

[36] D. M. Russinoff, 1989: Proteus: A Frame-Based Nonmonotonic Inference System. In *Object-Oriented Concepts, Databases, and Applications* edited by Won Kim and Frederick H. Lochovsky, ACM Press, New York, New York.

[37] N. Ballou, H. Chou, J. Garzar, W. Kim, C. Petrie, D. Russinoff, D. Steiner, and D. Woelk. 1988: Coupling an Expert System Shell with an Object-Oriented Database System. *Jour. of Object-Oriented Programming* (June/July).

[38] J.L. Encarnacao and P.C. Lockemann (Eds.), 1990: *Engineering Databases: Connecting Islands of Automation Through Databases*. New York: Springer-Verlag, pg. 133.

[39] S. Y. W. Su, 1986: Modeling Integrated Manufacturing Data with SAM*. *IEEE Computer* pp. 34-49.

[40] J. Rumbaugh, 1987: Relations as Semantic Constructs in an Object-Oriented Language. *OOPLAS'87* Proceedings, pp. 466-481.

[41] M.E.S. Loomis, A. Shah, and J.E. Rumbaugh, 1987: An Object Modeling Technique for Conceptual Design. *ECOOP'87*. Published as Lecture Notes in *Computer Science* 276 (New York: Springer-Verlag).

[42] W. Kim, E. Bertino and J. F. Garza, 1989: Composite Objects Revisited. *ACM SIGMOD* (March).

[43] R. J. Brachman and H. J. Levesque, 1987: Tales from the Far Side of KRYPTON, Lessons for Expert Database Systems from Knowledge Representation. In L. Kerschberg (Ed), *Expert Database Systems*, Menlo Park, CA: Benjamin/Cummings.

[44] R. Reiter and G. Criscuolo, 1980: Some Representational Issues in Default Reasoning. Technical Report No. 80-7, Dept. of Computer Science, University of British Columbia, Vancouver, B.C. (August).

[45] W. Cheng, M. Kifer, and D.S. Warren. 1988: Hilog as a Platform for Database Languages. In R. Hull, R. Morrison, and D. Stemple (Eds.) *Intl. Workshop on Database Programming Languages*, (June).

[46] W. Cheng, M. Dewitt, and D.S. Warren, 1989: Hilog· A First Order Semantics for Higher Order Logic Programming Constructs. In *Second International Workshop on Database Programming Languages*, Los Altos, CA: Morgan Kaufmann (June).

[47] S.S. Israni and P.J. Rafter, 1989: HP RTDB Boosts the Performance of Your Disk-Based Relational Database. In *HP Technical Report*, Hewlett-Packard Co., Palo Alto, CA.

[48] S. Tsur and C. Zaniolo, 1986: LDL: A Logic-Based Data Language. *MCC Technical Report* 026/86, Austin, Texas (February).

[49] T. Lunt and T. Garvey, 1990: Multilevel Security for Knowledge Based Systems. *Final Report, Project 7183*, SRI International, Menlo Park, CA.

MISSION

OF

ROME LABORATORY

Rome Laboratory plans and executes an interdisciplinary program in research, development, test, and technology transition in support of Air Force Command, Control, Communications and Intelligence (C^3I) activities for all Air Force platforms. It also executes selected acquisition programs in several areas of expertise. Technical and engineering support within areas of competence is provided to ESD Program Offices (POs) and other ESD elements to perform effective acquisition of C^3I systems. In addition, Rome Laboratory's technology supports other AFSC Product Divisions, the Air Force user community, and other DOD and non-DOD agencies. Rome Laboratory maintains technical competence and research programs in areas including, but not limited to, communications, command and control, battle management, intelligence information processing, computational sciences and software producibility, wide area surveillance/sensors, signal processing, solid state sciences, photonics, electromagnetic technology, superconductivity, and electronic reliability/maintainability and testability.